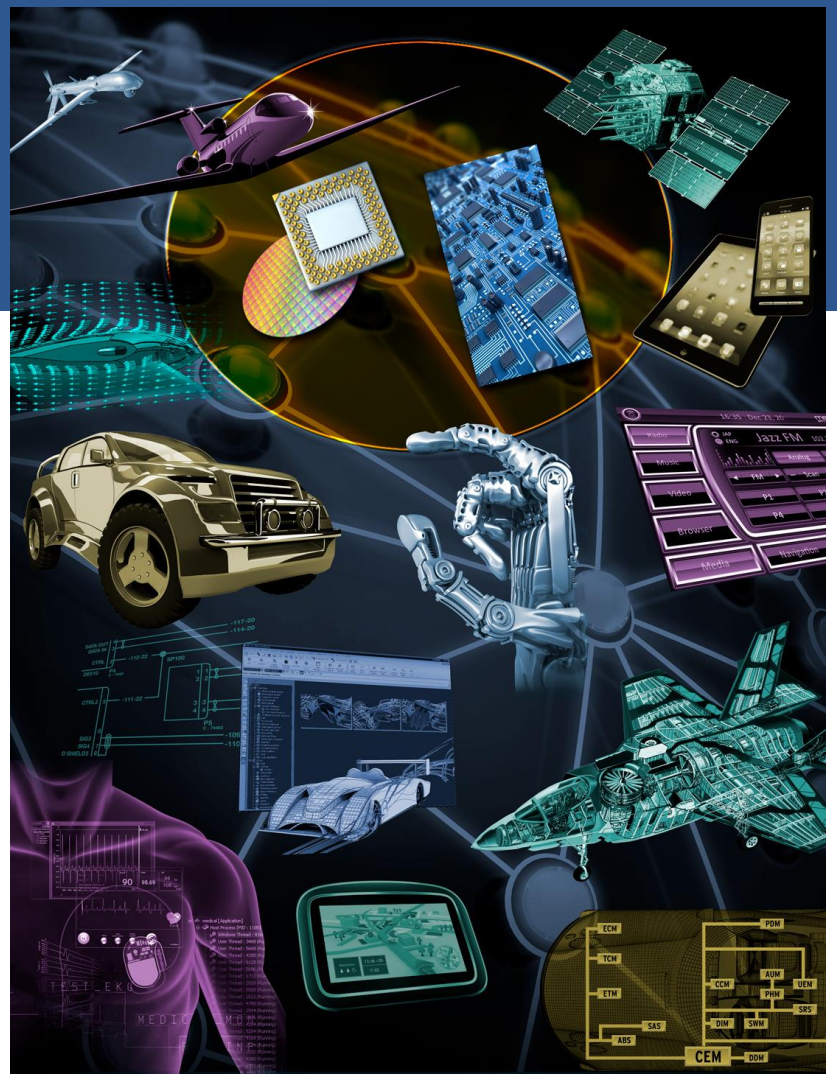


Automated Debugging of IJTAG Networks

Dr. Martin Keim

Engineering Director

Tessent Memory Test

Martin_Keim@Mentor.com

Nordic Test Forum, November 29, 2017

Mentor®
A Siemens Business

Outline

- Refresher on IJTAG (IEEE 1687)
- Debugging the network
  - What is in it?
  - Defects
  - Methodology
  - Visualization ?
- Summary

- Note:
  - This presentation shows work-in-progress
  - Feedback is very much appreciated

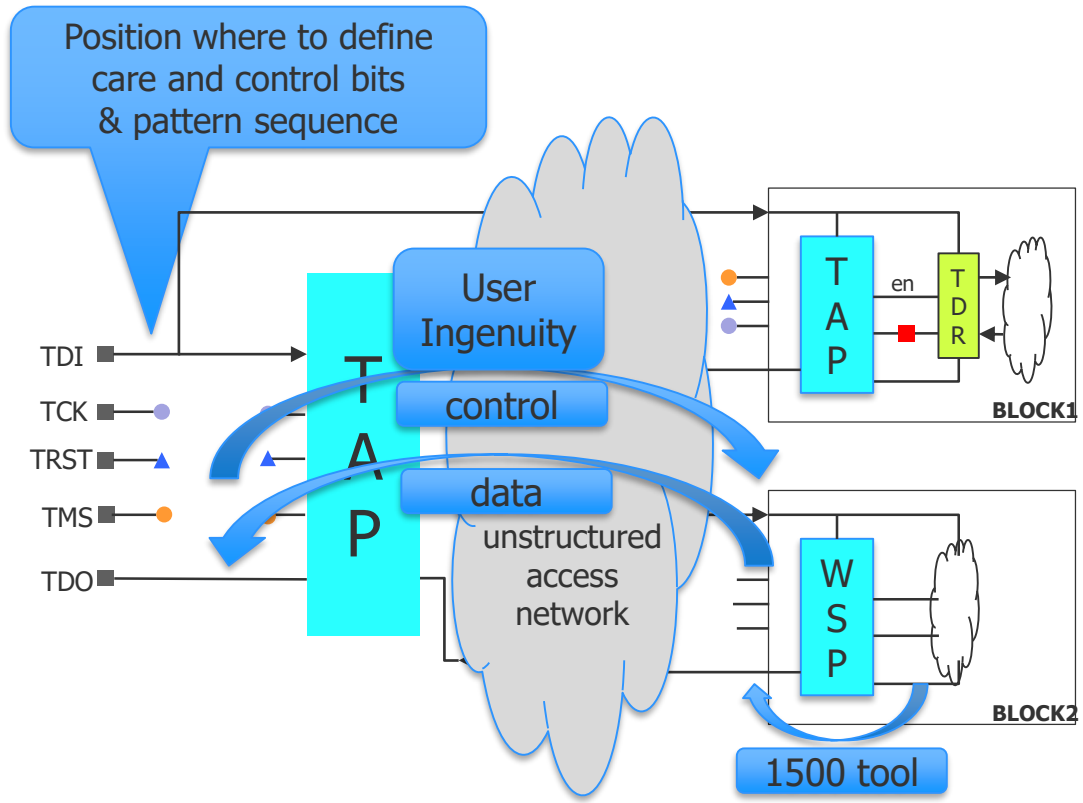
3 Approaches to IEEE 1687 (IJTAG)

- A business perspective
  - "I need to concentrate on my product's core value-add. Everything else should be commodity off the IP market."
- A design perspective
  - "I can no longer connect all IEEE 1149.1 IP to my TAP."
    - Note: IEEE 1149.1 in this historic context means pre-2013
- A DFT engineer's pain perspective
  - "Generating correct top level patterns for all my IP is painful and takes way too much time."
  - "ECO's are a nightmare"

3 Approaches to IEEE 1687 (IJTAG)

- A business perspective
  - "I need to concentrate on my product's core value-add. Everything else should be commodity off the IP market."
- A design perspective
  - "I can no longer connect all IEEE 1149.1 IP to my TAP."
    - Note: IEEE 1149.1 in this historic context means pre-2013
- A DFT engineer's pain perspective
  - "Generating correct top level patterns for all my IP is painful and takes way too much time."
  - "ECO's are a nightmare"

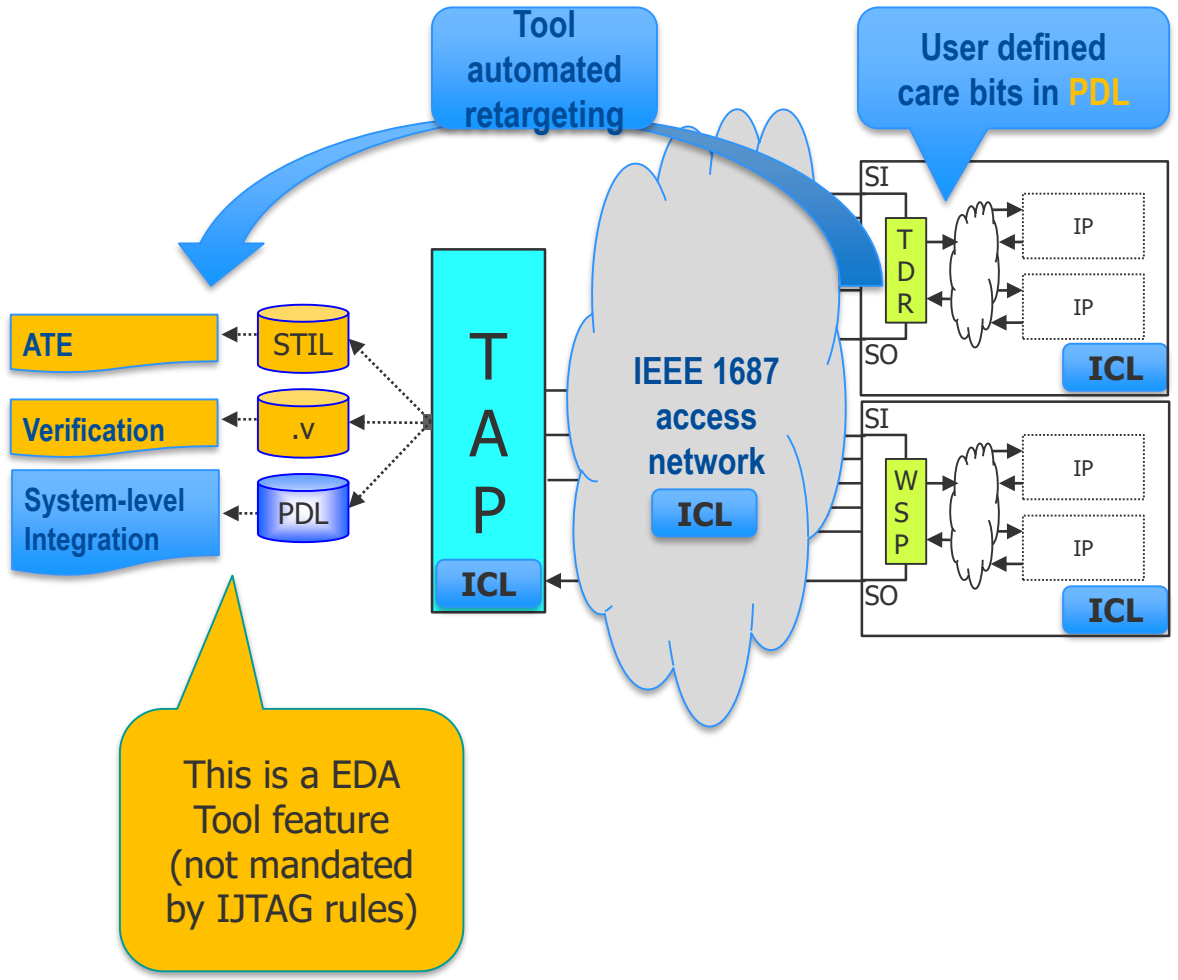
IEEE 1149.1 / 1500 Use Model



- Care bits are local to IP
- User must find top-level care & control bits
- (Semi-) manual process
- High potential for error
- Adding / changing IP causes ALL patterns to be invalid

- Huge time sink
- Test benches ???
- Diagnosis ???

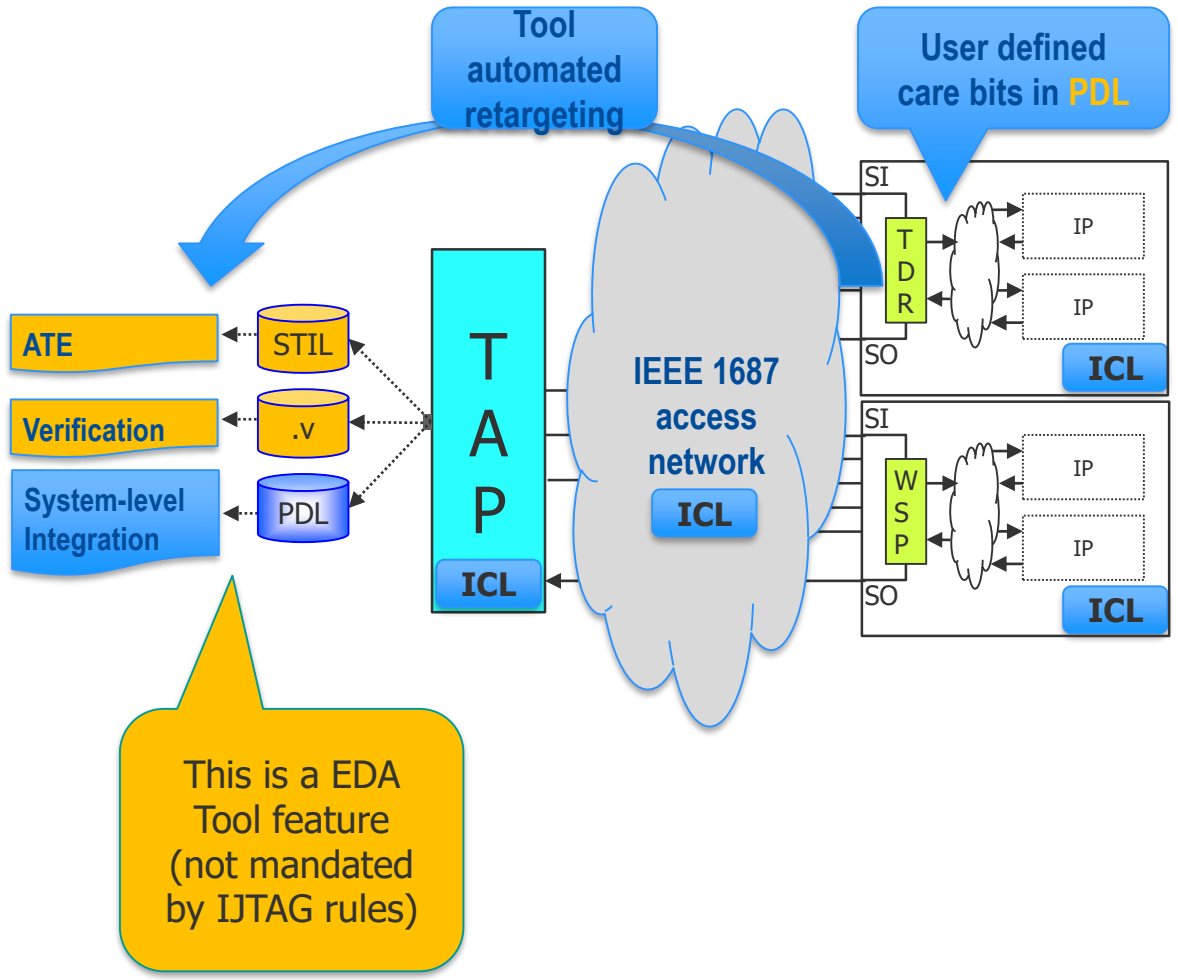
IEEE 1687 Use Model



IEEE 1687 defines:

- Procedural Description Language (**PDL**)
  - Describes IP usage at a given level
  - Facilitates automatic retargeting to any higher level
- Instrument Connectivity Language (**ICL**)
  - Describes only the (test) access / interface view of IP
  - Abstracts from IP implementation
  - Describes partial or complete networks
- Rules for 1687 IP realizations
  - Port functions, timing, connection
  - IJTAG does not require any new hardware
  - Follows 1149.1 hardware rules

IEEE 1687 and other future standards



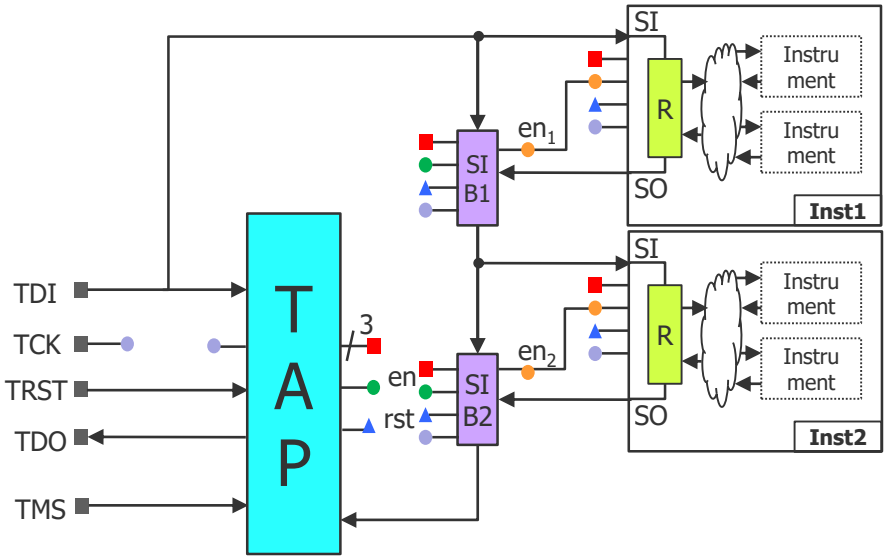
- P1687.1
  - Non-Tap chip-level interfaces
  - Example: SPI, I2C
- P1687.2
  - If the instruments are analog
- SJTAG working group
  - Connecting 1687, 1687.1 chips on a board
- P1838
  - Designs going 3D
  - Standard will describe only hardware.
  - HW and patterns can be modeled with 1687

An Implementation of PDL Retargeting

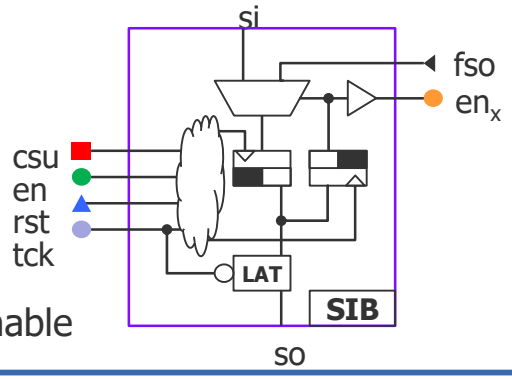
Retargeted PDL:
iWrite Tap.IR[2:0] 0b001
iApply

iWrite Sib1.S 0b1
iWrite Sib2.S 0b0
iApply

iWrite Inst1.R[3:0] 0b1111
iWrite Sib1.S 0b1
iWrite Sib2.S 0b0
iApply



User PDL:
iWrite Inst1.R 0xf
iApply



Does not show IP reuse

CSU = Capture, Shift, Update Enable

The depicted logic shall only serve as an example.
1687 does not require any particular implementation as long as the IO protocol is served correctly.

An Implementation of PDL Retargeting

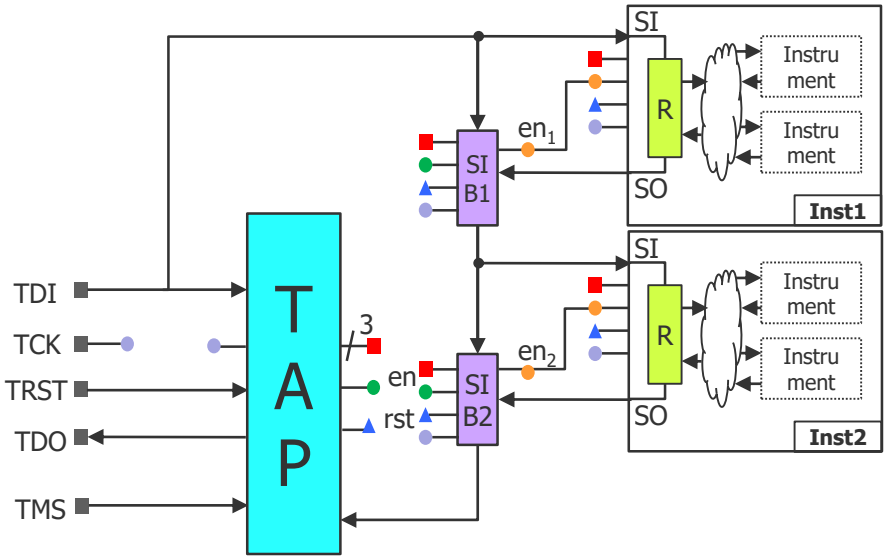
Retargeted PDL:

iWrite Tap.IR[2:0] 0b001
iApply

iWrite Sib1.S 0b1
iWrite Sib2.S 0b0
iApply

iWrite Inst1.R[3:0] 0b1111
iWrite Sib1.S 0b0
iWrite Sib2.S 0b1
iApply

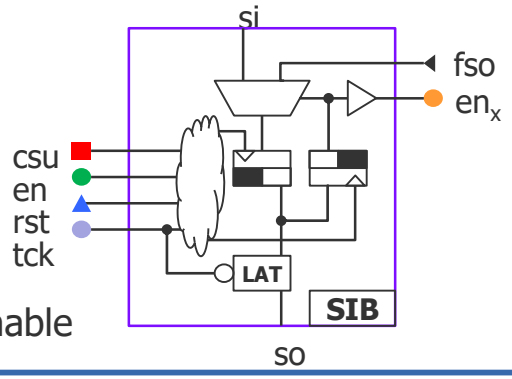
iWrite Sib1.S 0b0
iWrite Inst2.R[3:0] 0b1010
iWrite Sib2.S 0b1
iApply



User PDL:

iWrite Inst1.R 0xf
iApply

iWrite Inst2.R 0xa
iApply

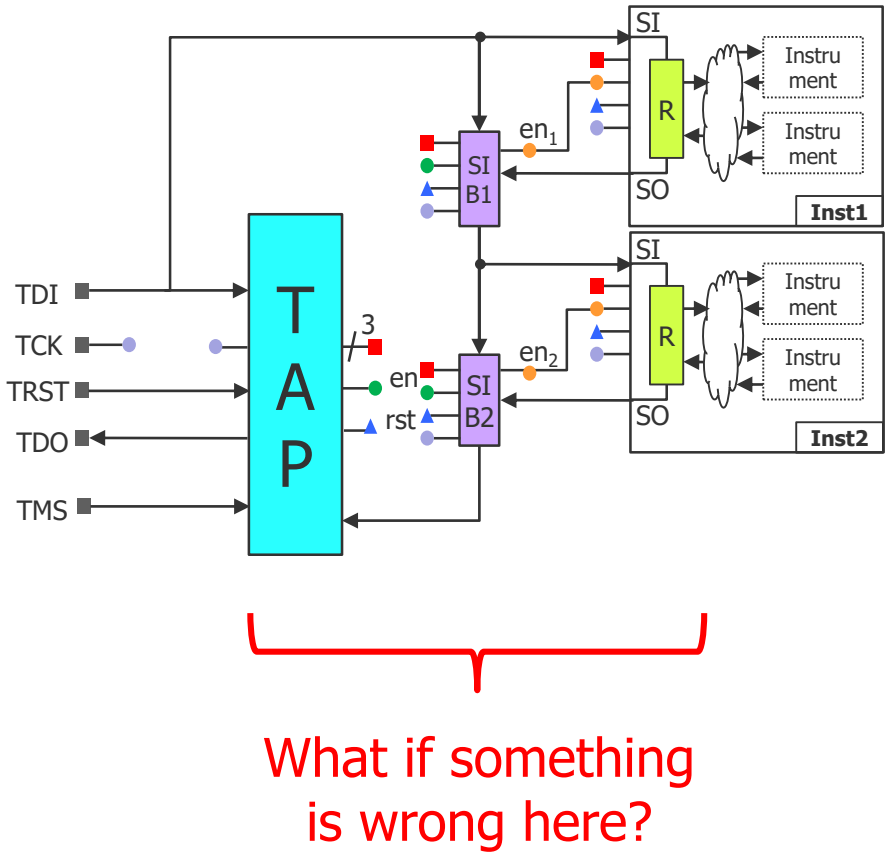


Does not show IP reuse

CSU = Capture, Shift, Update Enable

The depicted logic shall only serve as an example.
1687 does not require any particular implementation as long as the IO protocol is served correctly.

Problem Statement



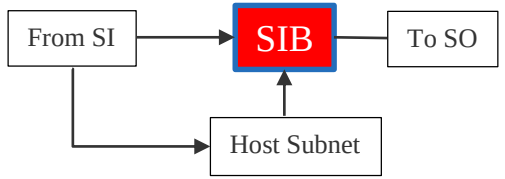
- The first test executed is ICLNetwork
- If the ICLNetwork test fails
  → no test/diagnosis of instruments may be possible/trusted
- As designs grow, so does the IJTAG network
  → higher probability for failures
- Currently don't have a way to diagnose or debug failures in IJTAG networks

- Questions to answer:
  - What components are in the IJTAG interconnect network?
  - What defects can we target?
  - How do we target these defects?
  - Can this be automated?

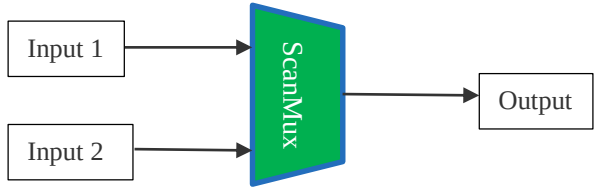
IJTAG Network Elements & Failure Modes

- Test Data Register (TDR)
  - Failure mode: bit stuck@1, bit stuck@0

- Segment Insertion Bit (SIB)
  - Built using a shift-update register and a Mux
  - Two (or more) inputs, one output
    - Deassert: pass data from SI to SO (client mode)
    - Assert: pass data from "Host subnet" (host mode)
  - Failure mode: stuck@assert, stuck@deassert

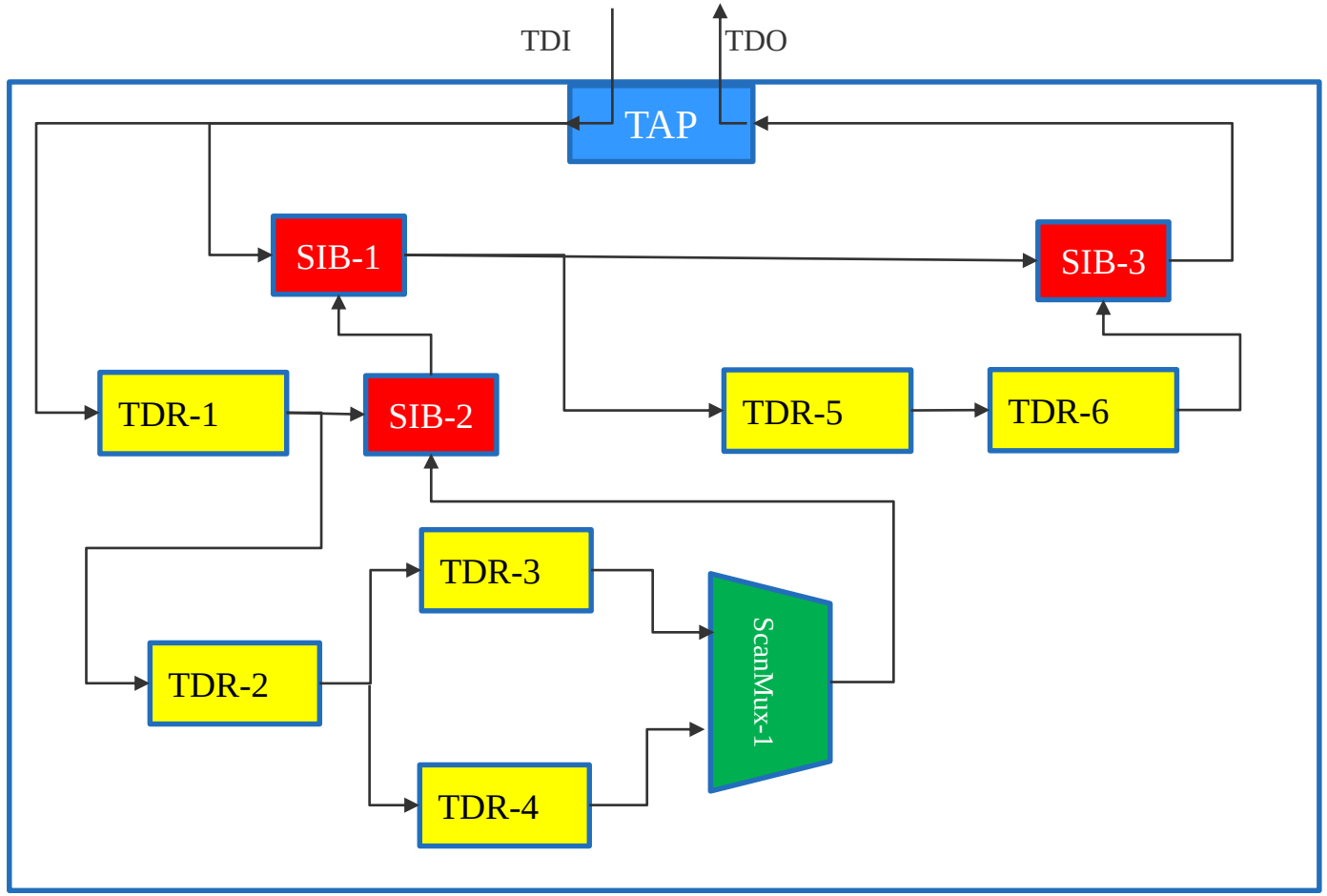


- ScanMux
  - Like SIB (built-in select register)
  - Failure mode: stuck@select0, stuck@select1

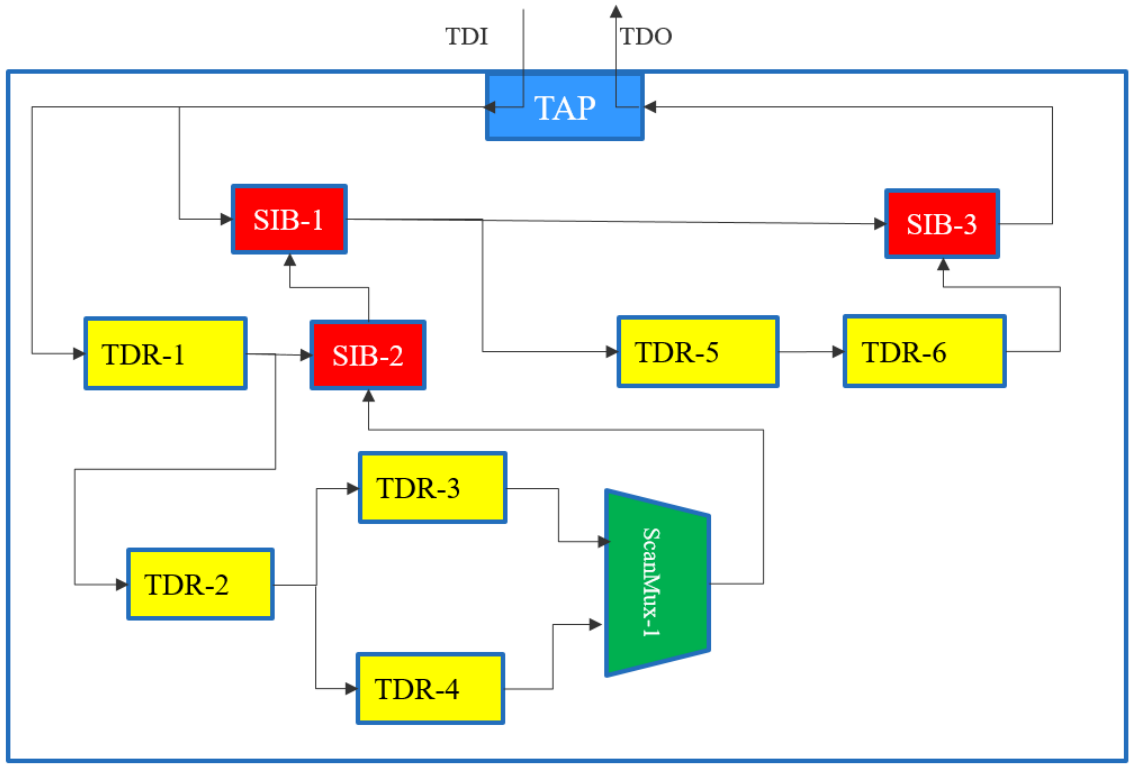


- Also considering slow-to-rise, slow-to-fall, …
  001100110011 → 000100010001 (just like ATPG scan chain diagnosis)

Example IJTAG Network



IJTAG Network Elements

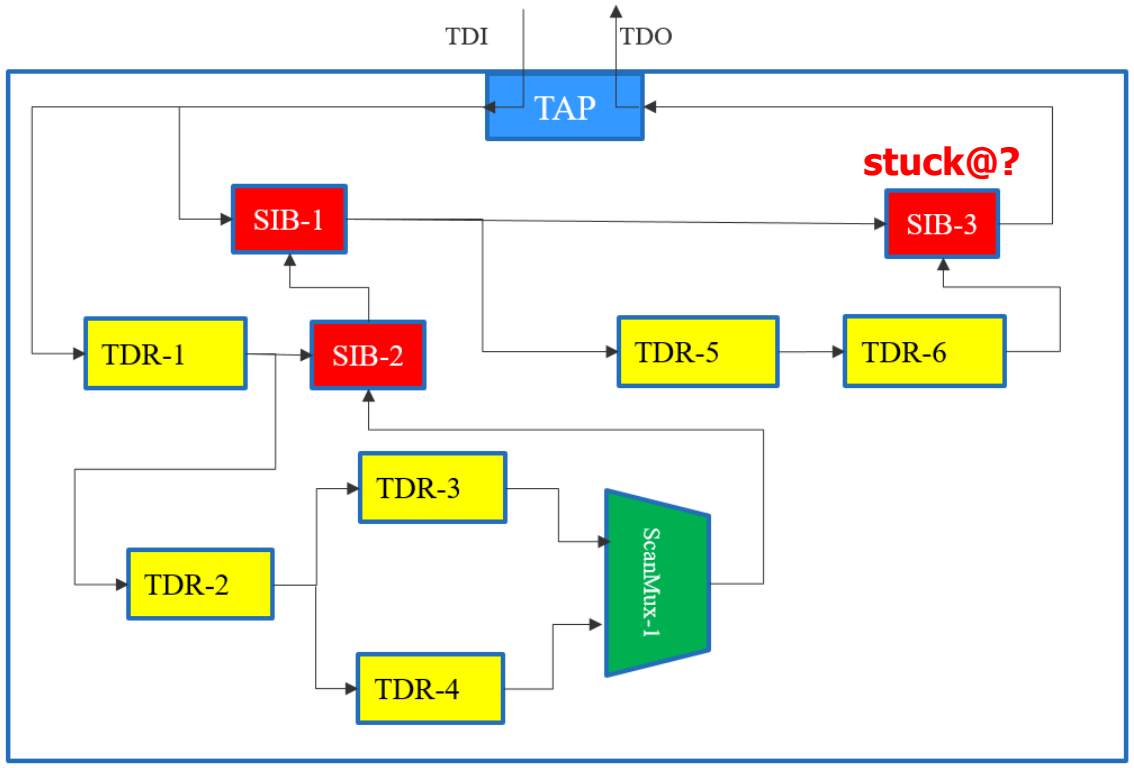


- Scan element
  - An instance of a SIB, ScanMux, or a TDR
- A scan path is an ordered sequence of scan elements
  - Starts from a scan primary input (i.e. TDI)
  - "Walks" a path through the network
  - Ends in a scan primary output (i.e. TDO)
- Valid scan path examples
  - SIB-1, SIB-3
  - SIB-1, TDR-5, TDR-6, SIB-3
  - TDR-1, TDR-2, TDR-4, ScanMux-1, SIB-2, SIB-1, SIB-3

One Technique: Over-Shifting

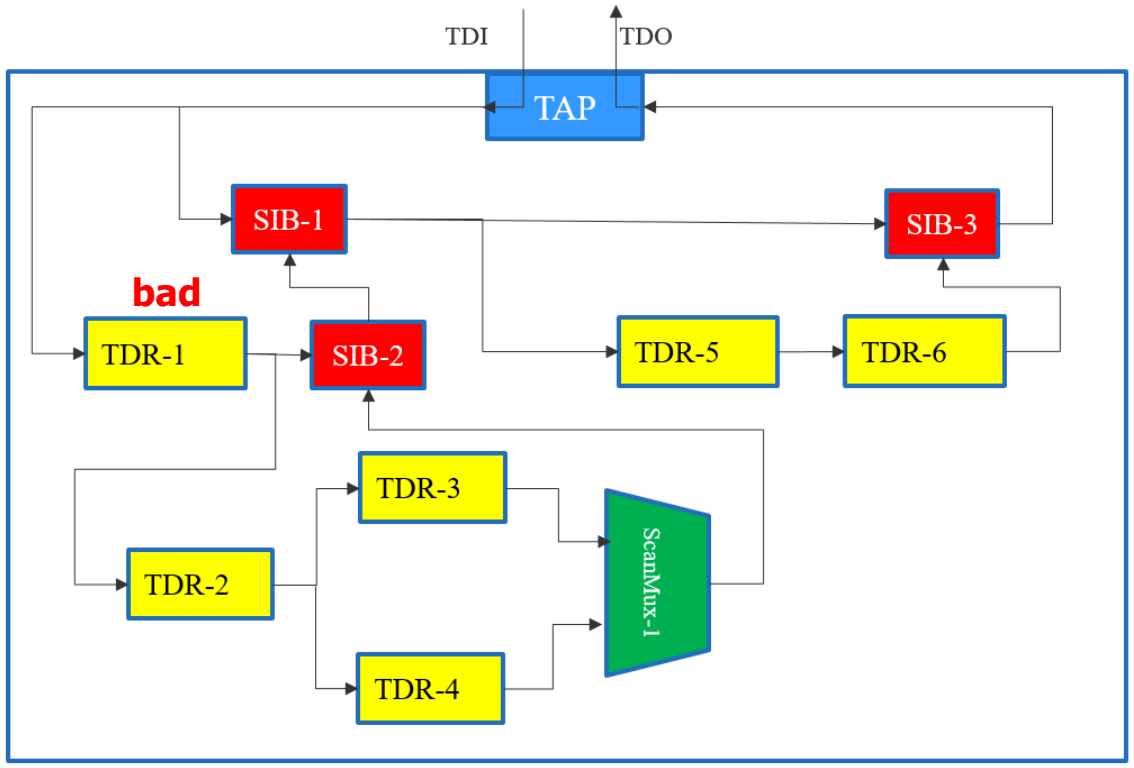
- Flush, i.e. scan-in, a bit pattern which is longer than the scan path length
- Look for that exact bit pattern in the scan-out data
- Usages
  - Check consistency of a register length between design and ICL
  - Diagnose a SIB or a ScanMux stuck at assert / deassert
- Not yet found a need for under-shifting

Over-Shifting Technique (Bad SIB-3 Example)



- Assume SIB-3 is bad
- Task: Determine stuck@assert or stuck@deassert
  - de-asserted scan path = (SIB-1@0, SIB-3@0)
  - asserted scan path = (SIB-1@0, TDR-5, TDR-6, SIB-3@1)
- If SIB-3 is stuck@deassert:
  - flush pattern 0011001100 will arrive at TDO too soon
- If SIB-3 is stuck@assert:
  - flush pattern 0011001100 will arrive at TDO too late
- Either of the above condition can be detected
  - flushing an appropriately long bit pattern 001100110011
  - and check when it appears at the TDO.
- If constant values are received
  → Bit failure on the path

Over-Shifting Technique (Bad TDR-1 Example)



- Assume TDR-1 is bad
- Task: Determine if it is bad due to length discrepancy between ICL and design.
- Minimum scan path = (TDR-1, SIB-2@0, SIB-1@1, SIB-3@0)
- If TDR-1 length in design < length in ICL
  - flush pattern 0011001100 will arrive at TDO too soon
- If TDR-1 length in design > length in ICL
  - flush pattern 0011001100 will arrive at TDO too late
- Either of the above condition can be detected
  - flushing an appropriately long bit pattern 001100110011
  - and check when it appears at the TDO.

Automatic Diagnosis

- Basic Hierarchical Algorithm
  - Methodical top down approach
  - Divide & conquer
  - New diagnosis patterns will be generated and executed as needed

Automatic Diagnosis (Hierarchical)

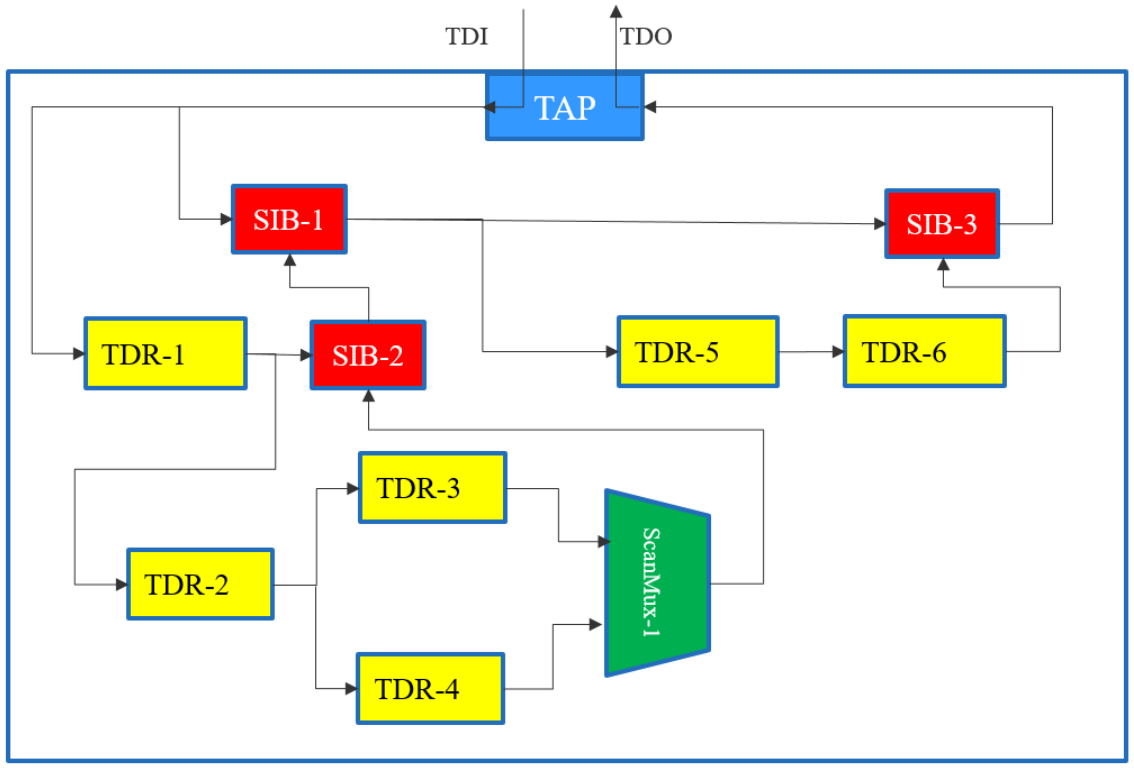
1. Call ***explore_ijtag_network*** to explore the top ring (i.e. all SIBS de-asserted)
2. Call ***setup_ijtag_scanpath*** to setup the scan path
3. Flush bit pattern using ***flush_ijtag_pattern*** to test it
4. If passed
   - Find a deeper ring (i.e. depth first)
   - Go to 2
5. If failed
   - Diagnose it
     - Flush various bit patterns
     - Use "over-shifting" technique for SIBs & ScanMuxes: Stuck at assert or de-assert
     - Use "over-shifting" technique for TDR's: Length consistent between design and ICL?
   - Find the next ring at the same level (don't go any deeper here)
   - Go to 2

Complexity:

Maximum Scan paths tested for 2 input SIBs & ScanMuxes = Number of SIBs + Number of ScanMuxes + 1

* Assuming SIBs and ScanMuxes all have 2 inputs

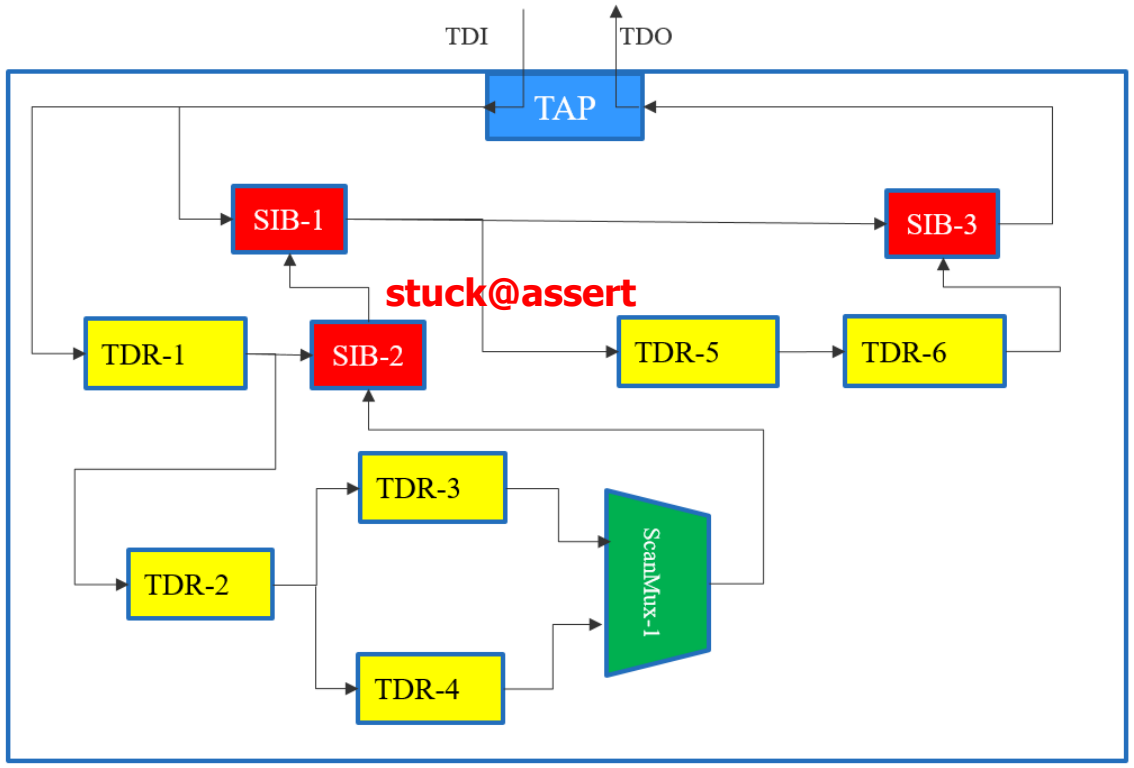
Hierarchical Algorithm Example



- For this example:
- Maximum number of explored scan paths = 3 + 1 + 1 = 5

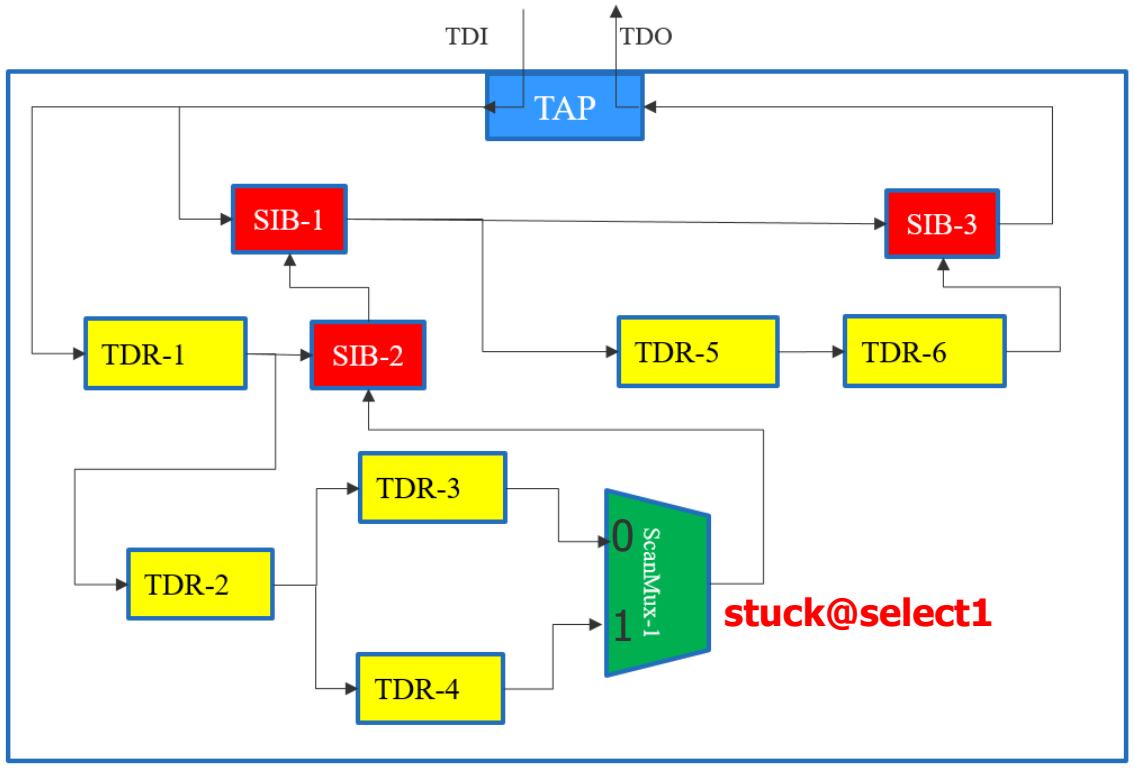
1. (SIB-1@0, SIB-3@0)
2. (TDR-1, SIB-2@0, SIB-1@1, SIB-3@0)
3. (TDR-1, TDR-2, TDR-3, ScanMux-1@0, SIB-2@1, SIB-1@1, SIB-3@0)
4. (TDR-1, TDR-2, TDR-4, ScanMux-1@1, SIB-2, SIB-1, SIB-3)
5. (SIB-1@0, TDR-5, TDR-6, SIB-3@1)

Hierarchical Algorithm Example 1



1. (SIB-1@0, SIB-3@0) - pass
2. (TDR-1, SIB-2@0, SIB-1@1, SIB-3@0) – fail
3. (TDR-1, TDR-2, TDR-3, ScanMux-1@0, SIB-2@1, SIB-1@1, SIB-3@0) - pass
4. "over-shifting" (TDR-1, SIB-2@0, SIB-1@1, SIB-3@0) – Look for the exact flush pattern at TDO
5. Conclude SIB-2 is stuck@assert

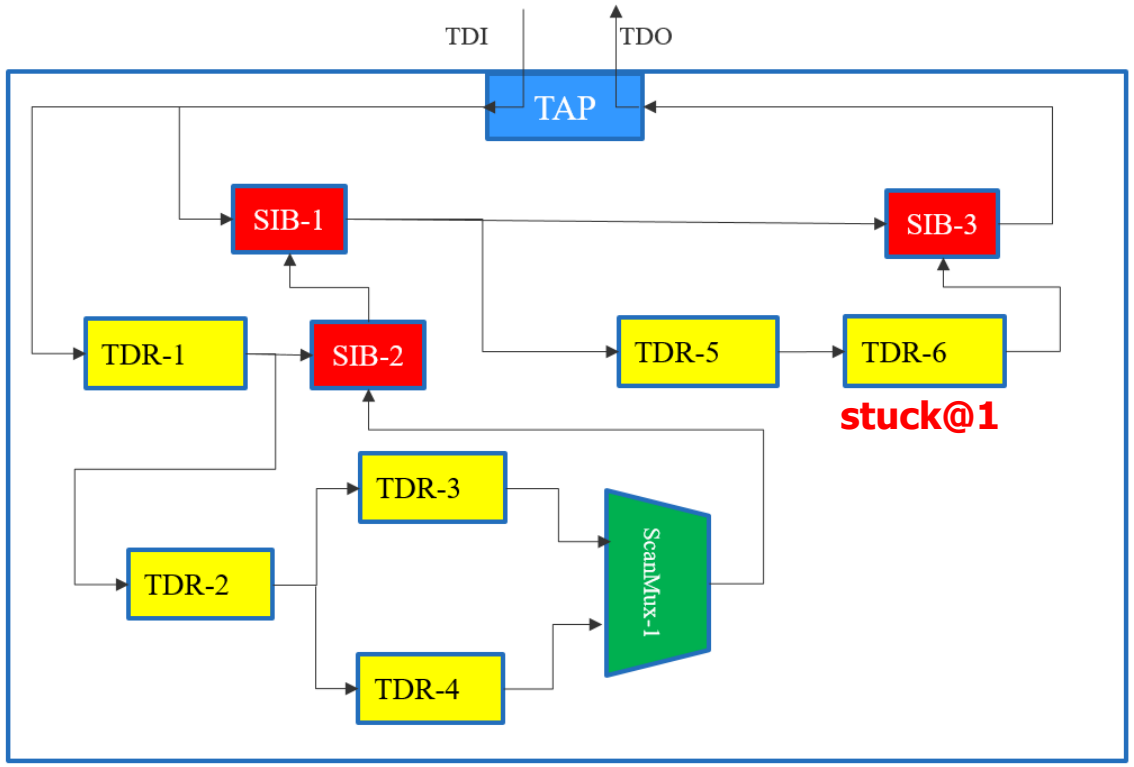
Hierarchical Algorithm Example 2



1. (SIB-1@0, SIB-3@0) - pass
2. (TDR-1, SIB-2@0, SIB-1@1, SIB-3@0) – pass
3. (TDR-1, TDR-2, TDR-3, ScanMux-1@0, SIB-2@1, SIB-1@1, SIB-3@0) - fail
4. (TDR-1, TDR-2, TDR-4, ScanMux-1@1, SIB-2@1, SIB-1@1, SIB-3@0) - pass
5. "over-shifting" (TDR-1, TDR-2, TDR-3, ScanMux-1@0, SIB-2@1, SIB-1@1, SIB-3@0) – observe exact flush pattern at TDO
6. Conclude ScanMux-1 is stuck@select1

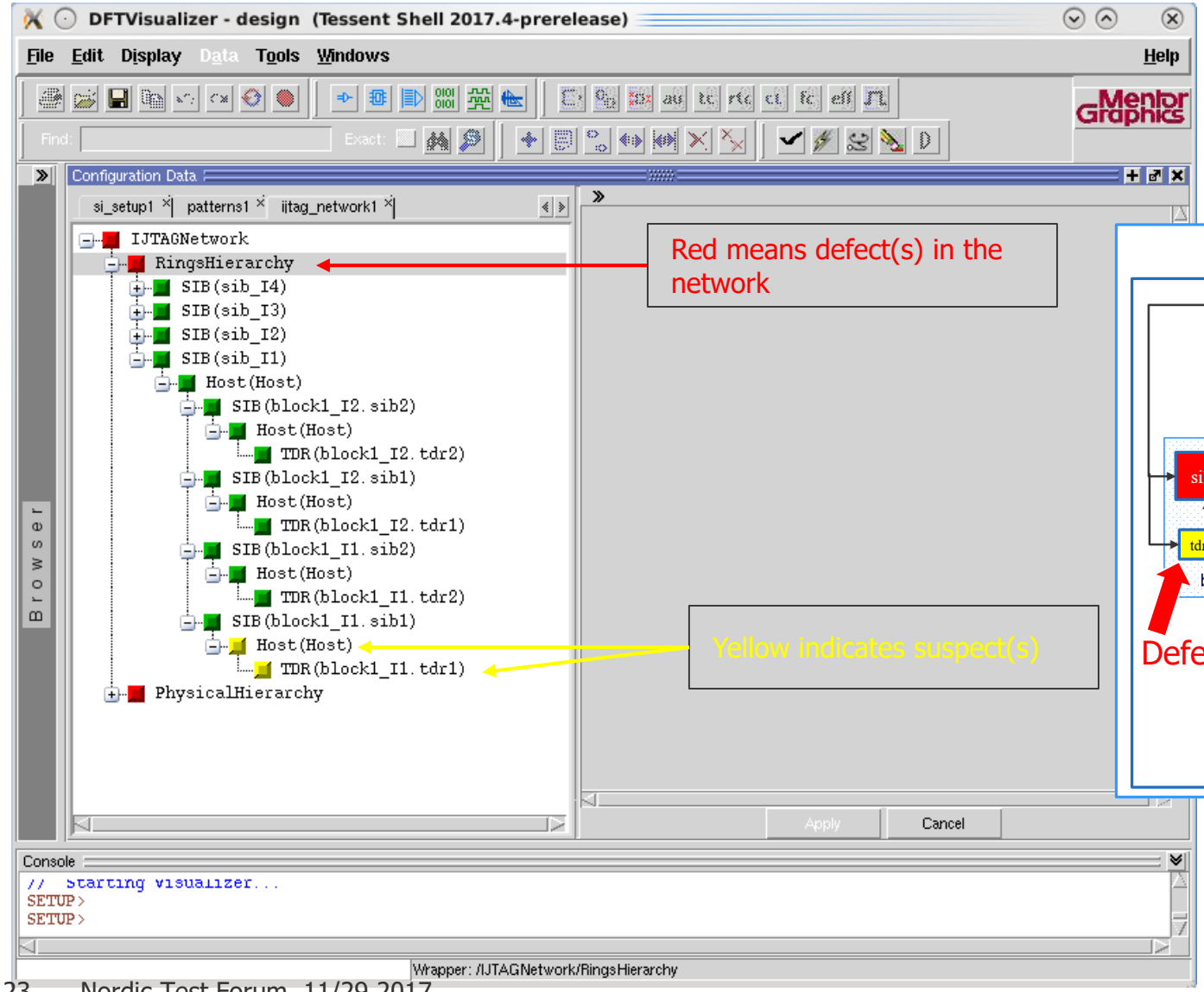
- If the length of TDR-3 is same as TDR-4
  - cannot discern if ScanMux-1 is stuck at select 0 or 1

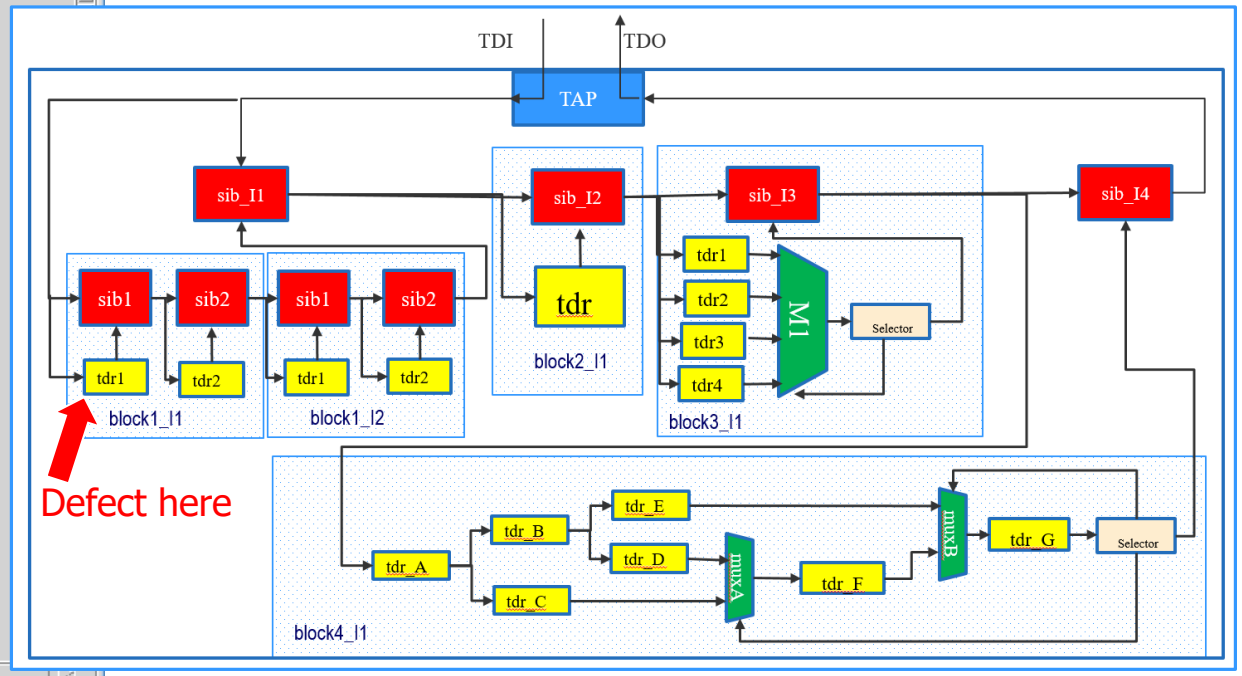
Hierarchical Algorithm Example 3



1. (SIB-1@0, SIB-3@0) - pass
2. (TDR-1, SIB-2@0, SIB-1@1, SIB-3@0) – pass
3. (TDR-1, TDR-2, TDR-3, ScanMux-1@0, SIB-2@1, SIB-1@1, SIB-3@0) - pass
4. (TDR-1, TDR-2, TDR-4, ScanMux-1@0, SIB-2@1, SIB-1@1, SIB-3@0) - pass
5. (SIB-1@0, TDR-5, TDR-6, SIB-3@1) – fail (all 1's)
6. "over-shifting" (SIB-1@0, TDR-5, TDR-6, SIB-3@1) – fail (all 1's)
7. Conclude TDR-5 and/or TDR-6 stuck at 1.

GUI Representation of the IJTAG Network





Summary

- Debugging the IJTAG Network will become important
- Prototype has been applied successfully at customers

- Can be done automatically
- Can be done interactively
- Can generate static ATE pattern set → offline diagnosis solution

- More ideas to implement
- Feedback is welcome

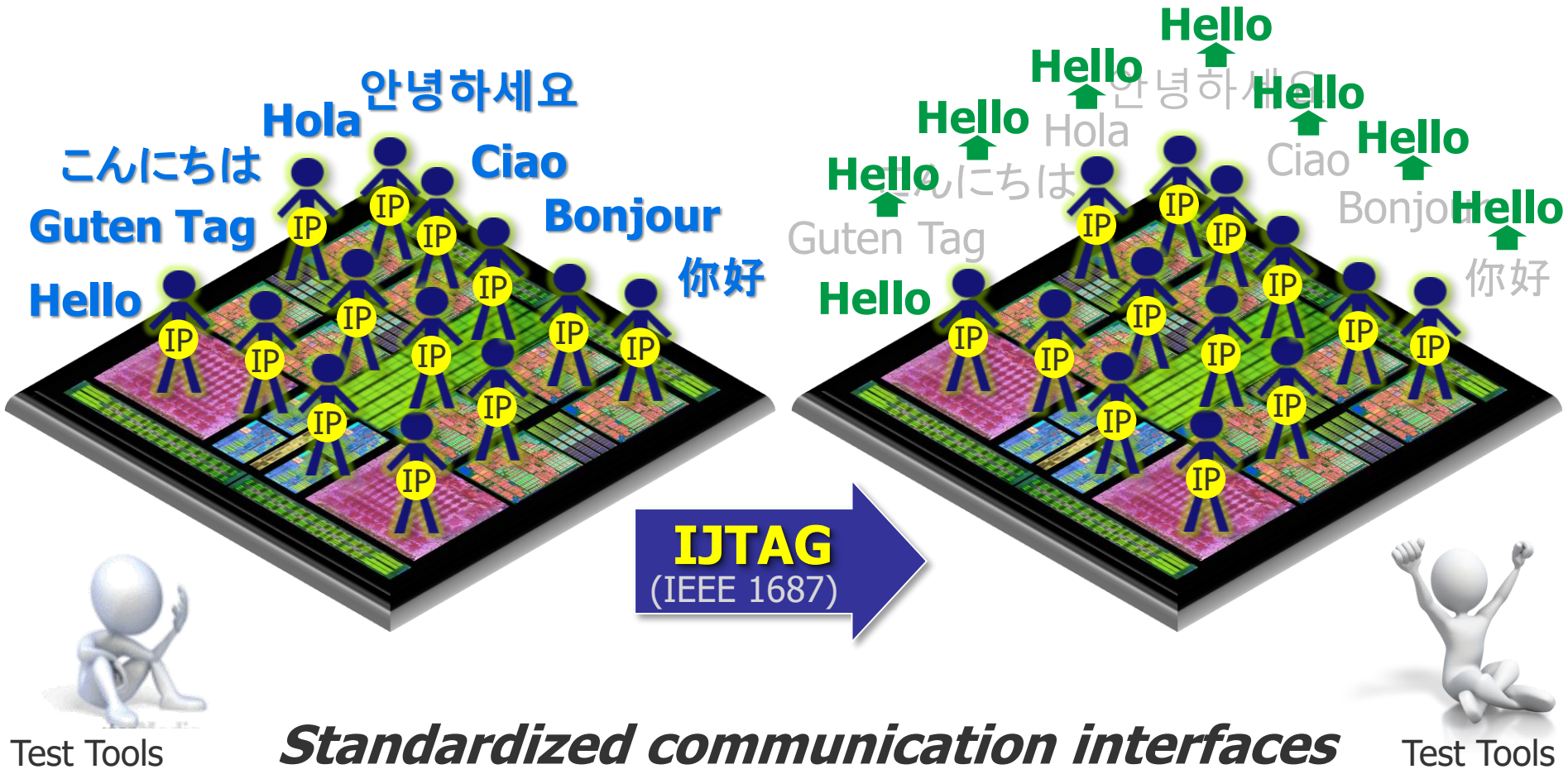
Q & A

BACKUP MATERIAL

3 Approaches to IEEE 1687 (IJTAG)

- A business perspective
  - "I need to concentrate on my product's core value-add. Everything else should be commodity off the IP market."
- A design perspective
  - "I can no longer connect all IEEE 1149.1 IP to my TAP."
    - Note: IEEE 1149.1 in this *historic* context means pre-2013
- A DFT engineer's pain perspective
  - "Generating correct top level patterns for all my IP is painful and takes way too much time."
  - "ECO's are a nightmare"

Business Problem: IP Diversity



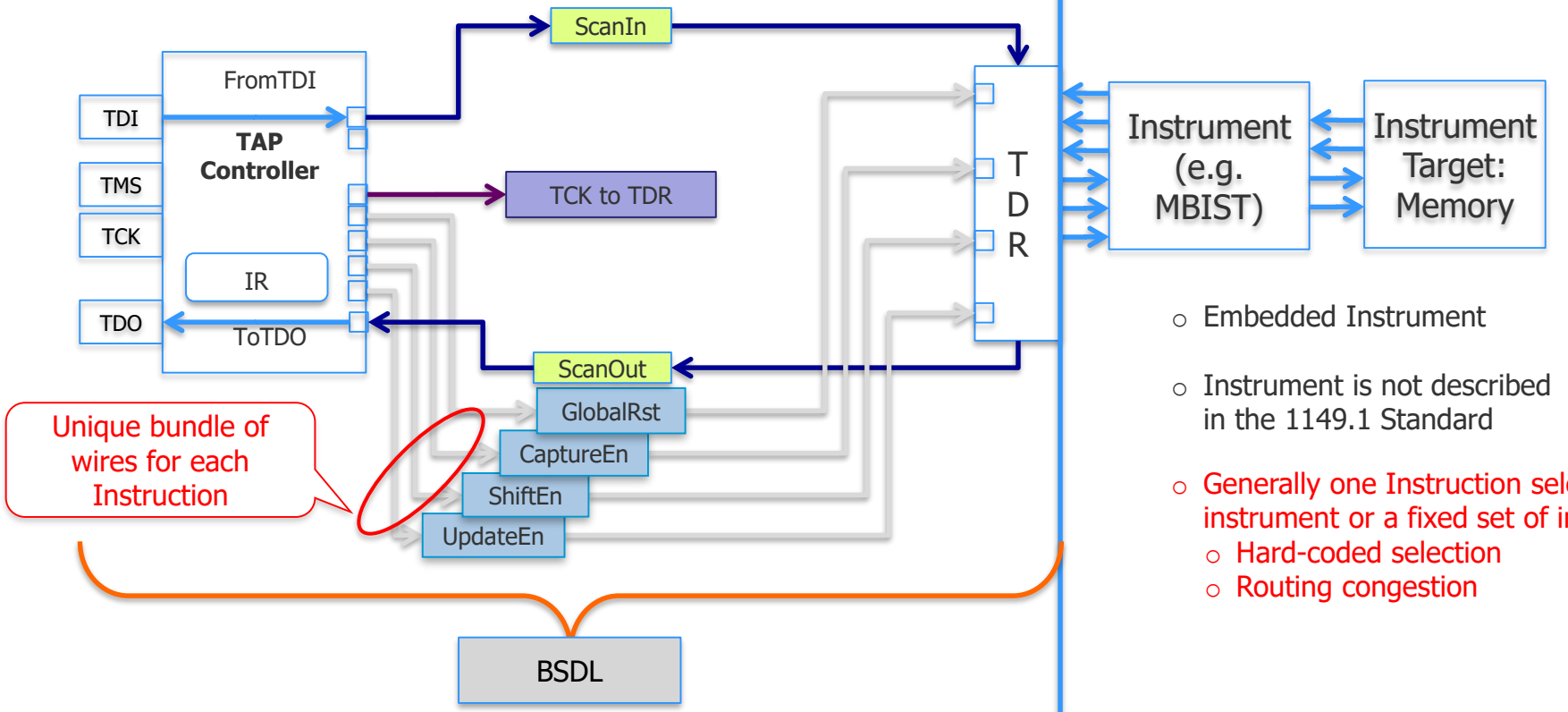
Standardized communication interfaces enable faster integration, test and debug

- flexible – scalable – non-intrusive – low cost -

3 Approaches to IEEE 1687 (IJTAG)

- A business perspective
  - "I need to concentrate on my product's core value-add. Everything else should be commodity off the IP market."
- A design perspective
  - "I can no longer connect all IEEE 1149.1 IP to my TAP."
    - Note: IEEE 1149.1 in this *historic* context means pre-2013
- A DFT engineer's pain perspective
  - "Generating correct top level patterns for all my IP is painful and takes way too much time."
  - "ECO's are a nightmare"

The 1149.1 Network with Instruments

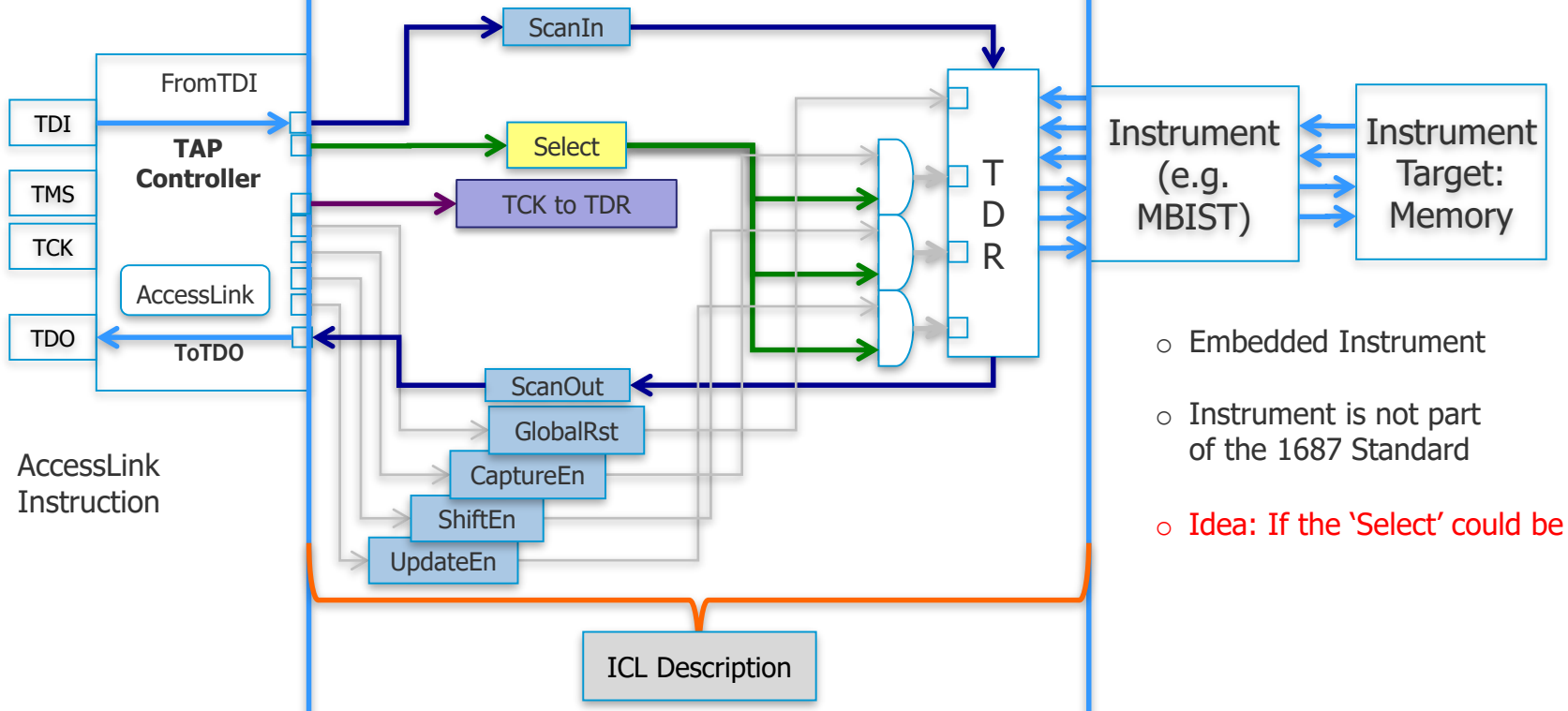


- The Scan Path Network is documented as a TDR with a length
- Decode for Operation Signals happens in the TAP Controller
- No Scan Path Management Features
- No description of the Instrument or its Vectors

- Embedded Instrument
- Instrument is not described in the 1149.1 Standard
- Generally one Instruction selects one instrument or a fixed set of instruments
  - Hard-coded selection
  - Routing congestion

JTAG Architecture | Instrument

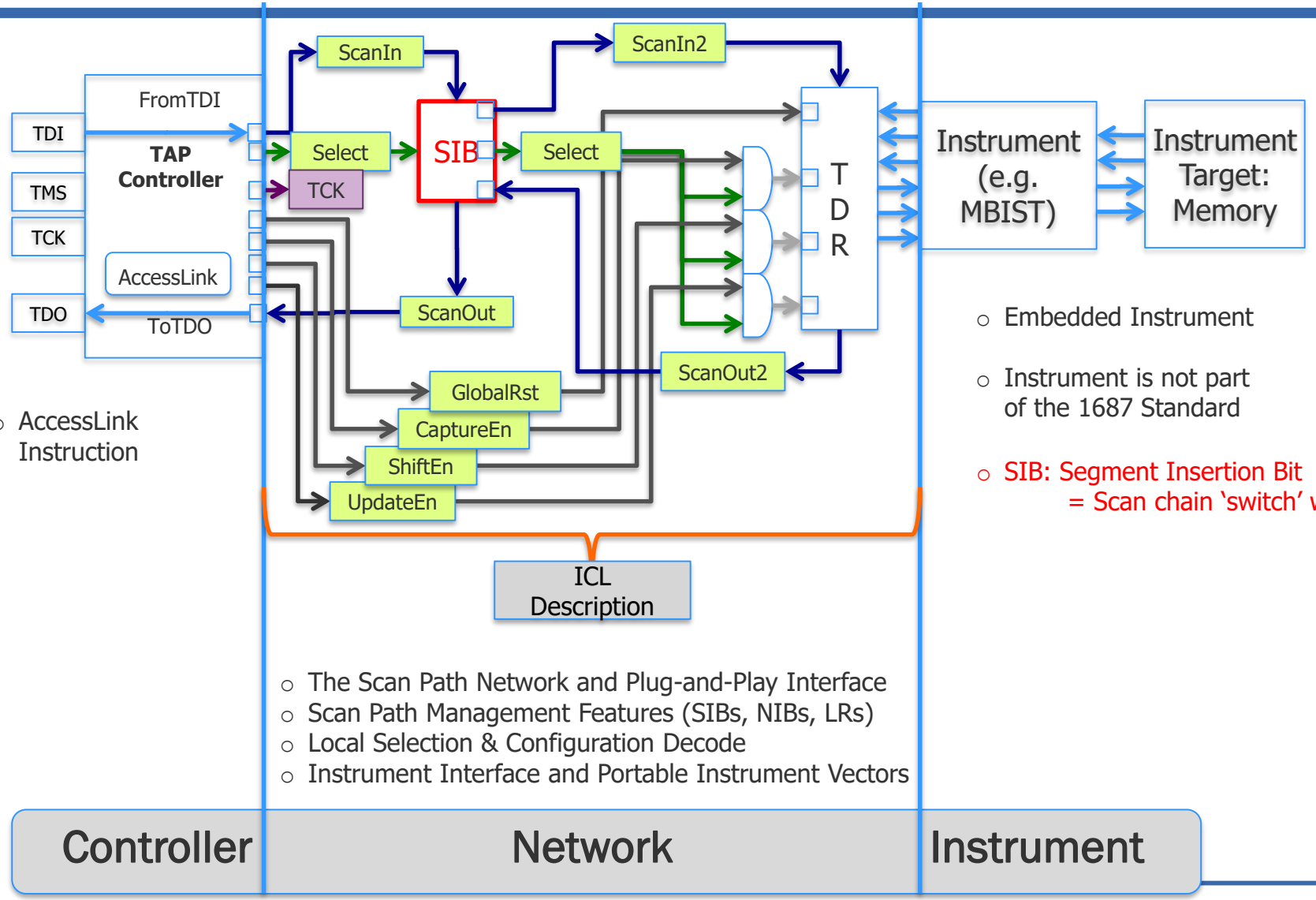
Basic 1687 Network with Instruments



- AccessLink Instruction

- Embedded Instrument
- Instrument is not part of the 1687 Standard
- Idea: If the 'Select' could be local, less routing needed

Basic 1687 Network with Instruments & SIB

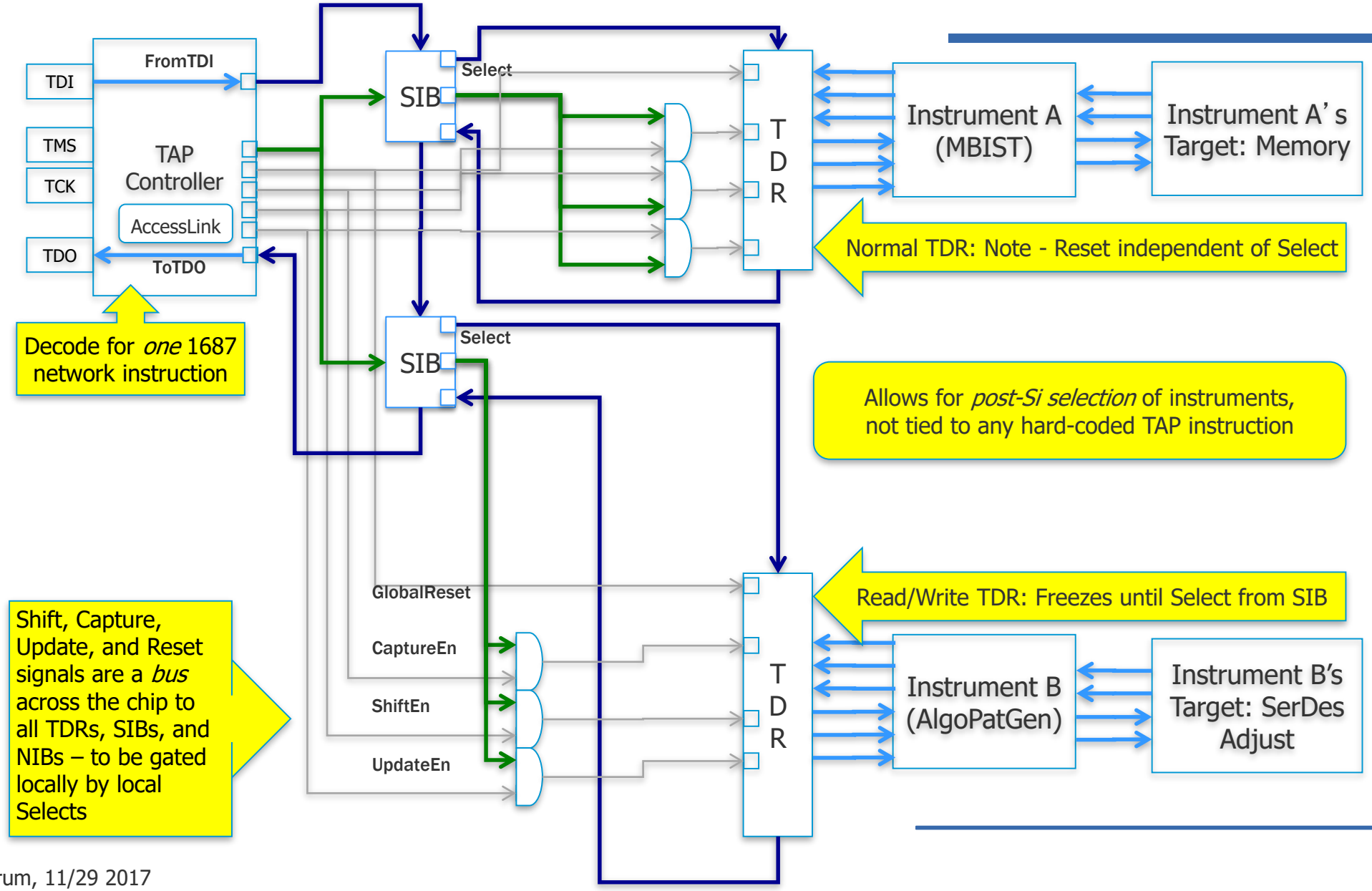


- AccessLink Instruction

- Embedded Instrument
- Instrument is not part of the 1687 Standard
- SIB: Segment Insertion Bit
  = Scan chain 'switch' with local select generator

- The Scan Path Network and Plug-and-Play Interface
- Scan Path Management Features (SIBs, NIBs, LRs)
- Local Selection & Configuration Decode
- Instrument Interface and Portable Instrument Vectors

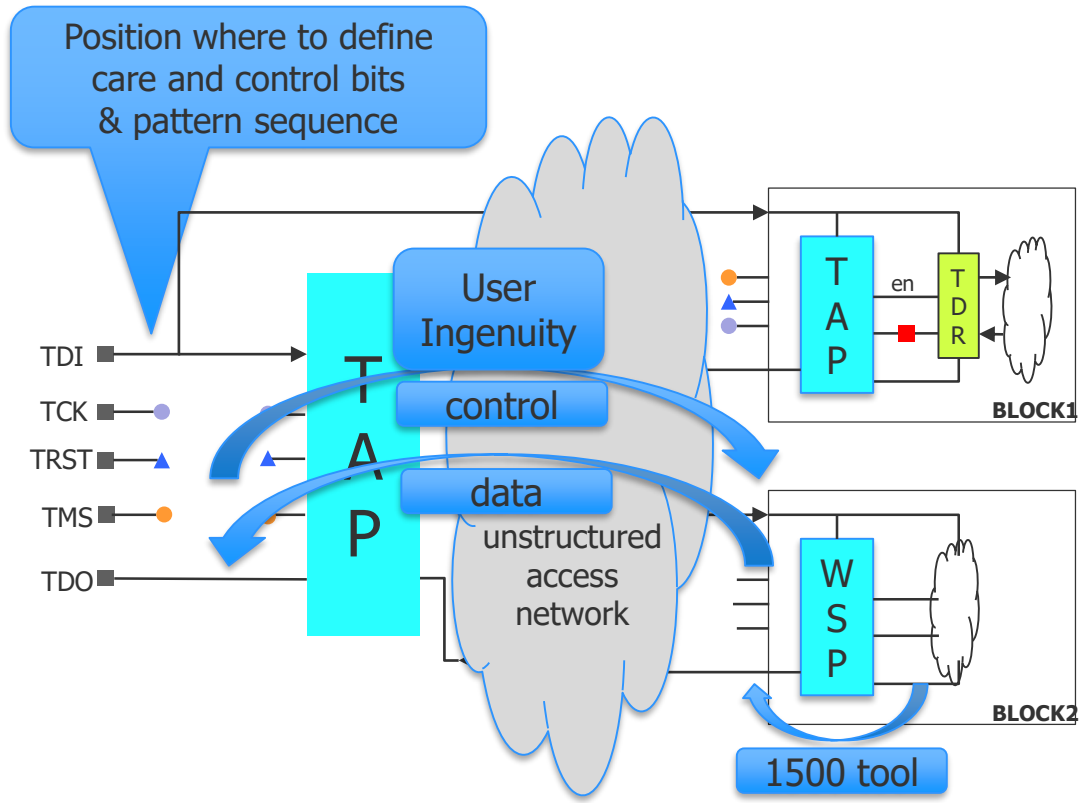
Complex 1687 Network with Instruments



3 Approaches to IEEE 1687 (IJTAG)

- A business perspective
  - "I need to concentrate on my product's core value-add. Everything else should be commodity off the IP market."
- A design perspective
  - "I can no longer connect all IEEE 1149.1 IP to my TAP."
    - Note: IEEE 1149.1 in this *historic* context means pre-2013
- A DFT engineer's pain perspective
  - "Generating correct top level patterns for all my IP is painful and takes way too much time."
  - "ECO's are a nightmare"

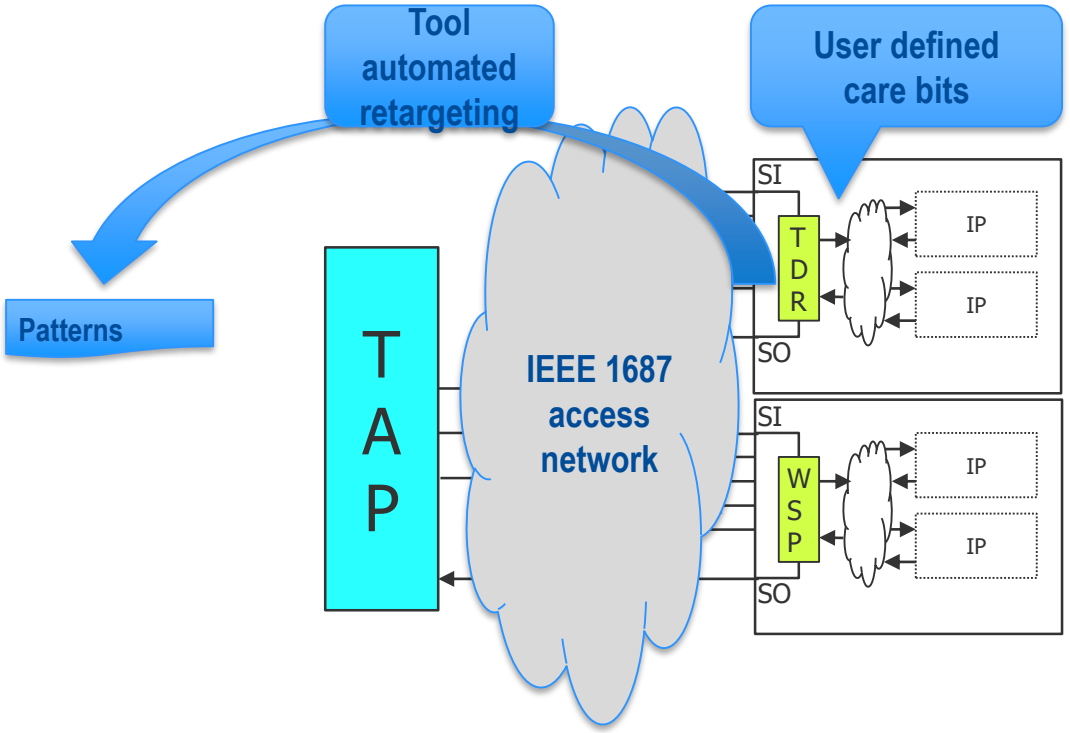
IEEE 1149.1 / 1500 Use Model

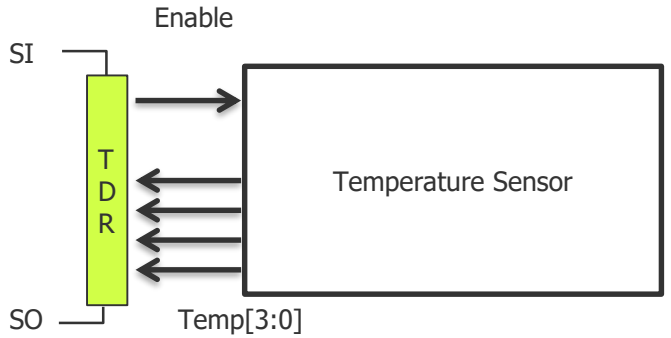


- Care bits are local to IP
- User must find top-level care & control bits
- (Semi-) manual process
- High potential for error
- Adding / changing IP causes ALL patterns to be invalid

- Huge time sink
- Test benches ???
- Diagnosis ???

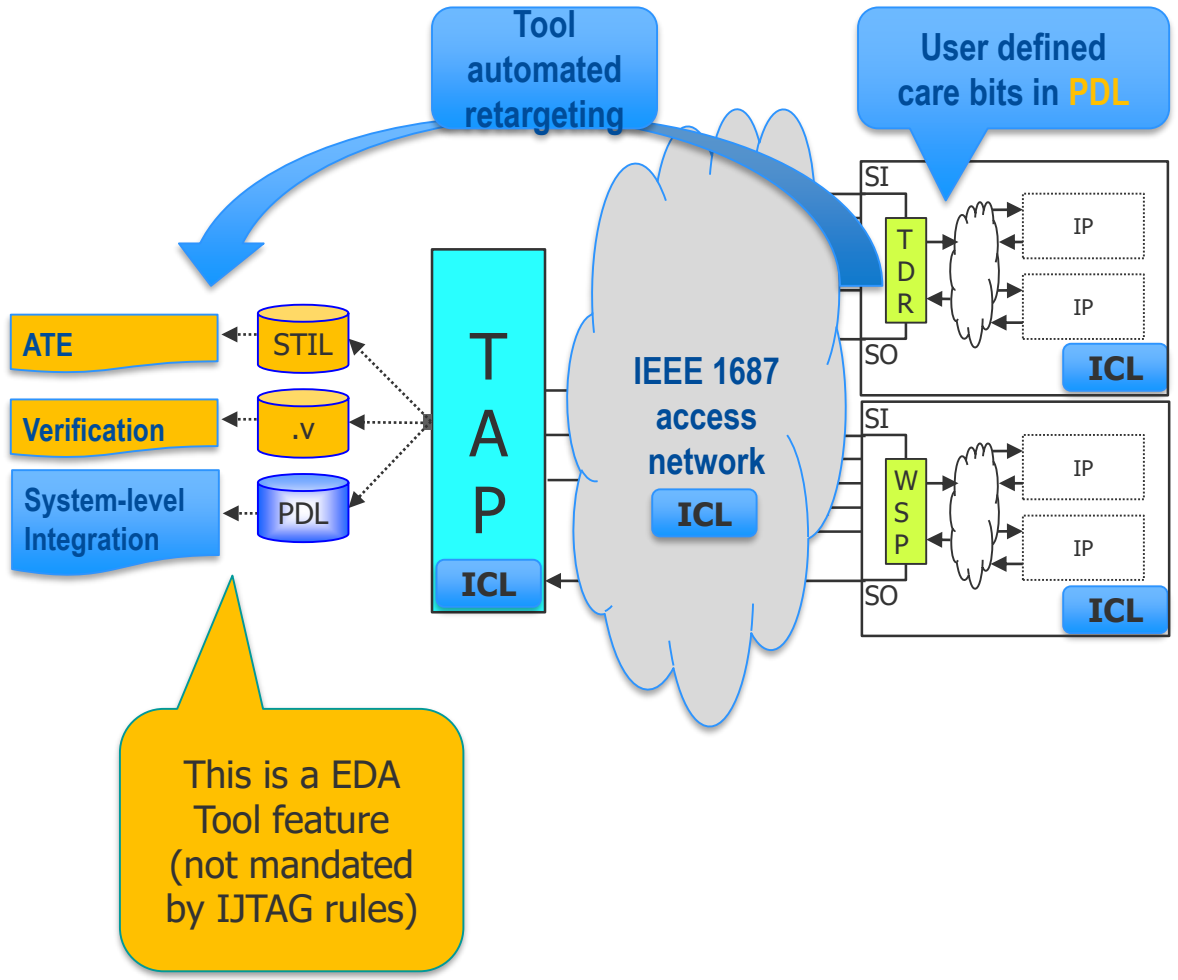
IEEE 1687 Use Model





- The process of moving IP-level patterns to higher design level is called '**retargeting**'
- IJTAG allows *automation* by an EDA tool

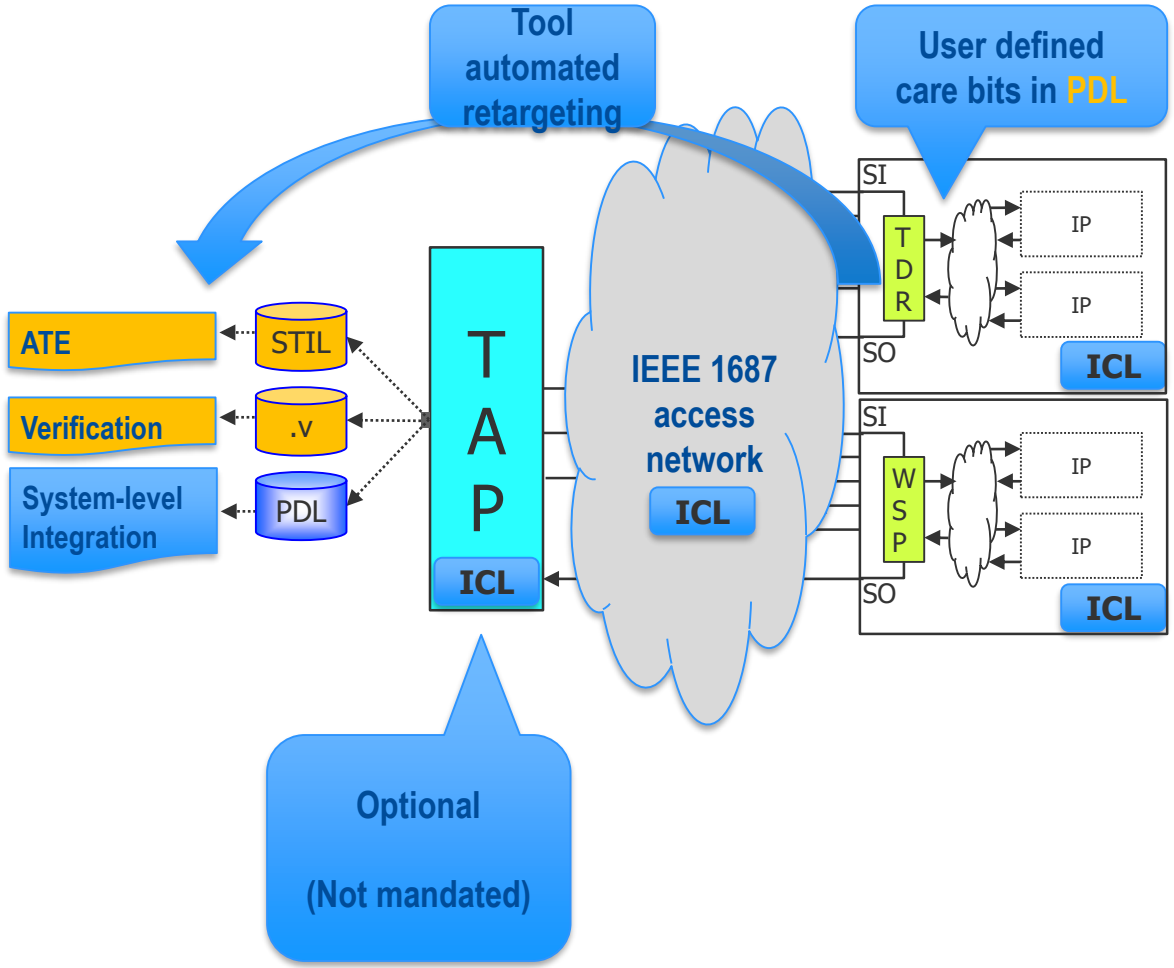
IEEE 1687 Use Model



IEEE 1687 defines:

- Procedural Description Language (**PDL**)
  - Describes IP usage at a given level
  - Facilitates automatic retargeting to any higher level
- Instrument Connectivity Language (**ICL**)
  - Describes only the (test) access / interface view of IP
  - Abstracts from IP implementation
  - Describes partial or complete networks
- Rules for 1687 IP realizations
  - Port functions, timing, connection
  - IJTAG does not require any new hardware
  - Follows 1149.1 hardware rules

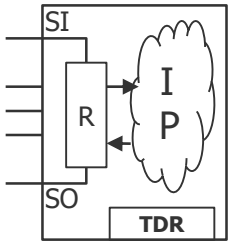
IEEE 1687 Use Model



Instrument Interfaces

- Serial access
  - Follows 1149.1 protocol
  - TDR, WSP, TAP, etc.
- Parallel access:
  - Read/write from data ports
  - Parallel scan chains (TDRs)
- Bus access:
  - Supports read/write enable
  - Address/Data
  - PDL implements protocol
- Same applies to top level access:
  - Serial (with/without TAP)
  - Parallel
  - Bus

Example 1: A Test Data Register Controlled IP




```
Module TDR {
   ScanInPort        SI ;
   ScanOutPort       SO { Source R[0] ; }
   ShiftEnPort       SE ;
   CaptureEnPort     CE ;
   UpdateEnPort      UE ;
   SelectPort        EN ;
   TCKPort           TCK;

   ScanRegister R[7:0] {
      ScanInSource    SI ;
   }
}
```

ICL

- Ports have semantic
- Relative timing of events at ports is defined by 1687
- Objects like registers have built-in properties
- The IP itself is not modeled

```
iProcsForModule TDR

iProc write_to_tdr { value } {
   iNote "Writing '$value' to register R of module TDR"
   iWrite R $value
   iApply
}

iProc run_testX { } {
   iCall write_to_tdr 0b10010110
   iRunLoop 1000 -tck
   iRead R 0xff
   iApply
}
```

PDL

- Looks like TCL
- Procedures are bound to modules
- Reading and writing only of care bits
- 1687 tool adds control bits
- 1687 tool takes care of the execution

Example 1: A Test Data Register Controlled IP

- PDL is not a pattern description in the sense of STIL, etc.
- PDL is a command language!
- PDL instructs an EDA tool how to compute patterns through retargeting

```
iProcsForModule TDR

iProc write_to_tdr { value } {
   iNote "Writing '$value' to register R of module TDR"
   iWrite R $value
   iApply
}

iProc run_testX { } {
   iCall write_to_tdr 0b10010110
   iRunLoop 1000 -tck
   iRead R 0xff
   iApply
}
```

PDL

- Looks like TCL
- Procedures are bound to modules
- Reading and writing only of care bits
- 1687 tool adds control bits
- 1687 tool takes care of the execution